Burn

Generated by Doxygen 1.8.11

Contents

Chapter 1

burn

Daemon running on Raspberry Pi (archlinuxarm), controlling a Canberra Osprey gamma detector and a G-Star IV GPS device.

Acquisitions and measurements are merged, saved to disk and optionally transferred over a TCP connection to the controlling application, crash.

Dependencies:

- Canberra Osprey SDK V1.0.1

This software is part of a drone project at Norwegian Radiation Protection Authority (NRPA)

Chapter 2

Namespace Index

2.1 Packages

Here are the packages with brief descriptions (if available):

Chapter 3

Hierarchical Index

3.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

Chapter 4

Class Index

4.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 5

File Index

5.1 File List

Here is a list of all files with brief descriptions:

Chapter 6

Namespace Documentation

6.1 burn Namespace Reference

Namespaces

- burn
- helpers
- net_proc
- proto
- spec_proc

6.2 burn.burn Namespace Reference

Classes

- class Burn

Variables

- filename
- level

6.2.1 Variable Documentation

6.2.1.1 burn.burn.filename

Definition at line 125 of file burn.py.

6.2.1.2 burn.burn.level

Definition at line 125 of file burn.py.

6.3 burn.helpers Namespace Reference

Functions

- def setblocking (fd, state)

6.3.1 Function Documentation

6.3.1.1 def burn.helpers.setblocking (*fd*, *state*)

```
Set the blocking state of a file descriptor
```

Definition at line 22 of file helpers.py.

```
def setblocking(fd, state):
    """
    Set the blocking state of a file descriptor
    """
    flags = fcntl.fcntl(fd, fcntl.F_GETFL)
    if state:
        fcntl.fcntl(fd, fcntl.F_SETFL, flags & ~os.O_NONBLOCK)
    else:
        fcntl.fcntl(fd, fcntl.F_SETFL, flags | os.O_NONBLOCK)

```

6.4 burn.net_proc Namespace Reference

Classes

- class NetProc

Variables

- string HOST = ''
- int PORT = 7000

6.4.1 Variable Documentation

6.4.1.1 string burn.net_proc.HOST = ''

Definition at line 25 of file net_proc.py.

6.4.1.2 int burn.net_proc.PORT = 7000

Definition at line 26 of file net_proc.py.

6.5 burn.proto Namespace Reference

Classes

- class Message

6.6 burn.spec_proc Namespace Reference

Classes

- class GpsThread
- class SessionThread
- class SpecProc

Chapter 7

Class Documentation

7.1 burn.burn.Burn Class Reference

Public Member Functions

- def __init__ (self)
- def run (self)
- def dispatch_net_msg (self, msg)
- def dispatch_spec_msg (self, msg)
- def __enter__ (self)
- def __exit__ (self, exc_type, exc_value, traceback)

Public Attributes

- running
- fds
- fdn
- s
- n

7.1.1 Detailed Description

Definition at line 30 of file burn.py.

7.1.2 Constructor & Destructor Documentation

7.1.2.1 def burn.burn.Burn.__init__ (*self*)

```
initialize main controller
```

Definition at line 32 of file burn.py.

```
    def __init__(self):
        """
        initialize main controller
        """
        self.running = False

        # Create pipes for message passing between gps_proc and spec_proc
        fds_pass, self.fds = Pipe()
        fdn_pass, self.fdn = Pipe()

        # Make file descriptors non-blocking
        setblocking(self.fds, 0)
        setblocking(self.fdn, 0)

        # Create and start child processes
        self.s = SpecProc(fds_pass)
        self.n = NetProc(fdn_pass)
        self.s.start()
        self.n.start()

        # Close unused file descriptors
        # The child processes inherits *all* parent descriptors (FIXME)
        fds_pass.close()
        fdn_pass.close()

```

7.1.3 Member Function Documentation

7.1.3.1 def burn.burn.Burn.__enter__ (*self*)

Definition at line 106 of file burn.py.

```
    def __enter__(self):
        return self

```

7.1.3.2 def burn.burn.Burn.__exit__ (*self,* *exc_type,* *exc_value,* *traceback*)

Definition at line 109 of file burn.py.

```
    def __exit__(self, exc_type, exc_value, traceback):
        # Main controller destructor, cleaning up
        self.fds.close()
        self.fdn.close()

        self.s.join()
        self.n.join()

        logging.info('ctrl: terminating')

```

7.1.3.3 def burn.burn.Burn.dispatch_net_msg (*self,* *msg*)

```
Function to handle messages from the network
```

Definition at line 81 of file burn.py.

```
    def dispatch_net_msg(self, msg):
        """
        Function to handle messages from the network
        """
        if not msg:
            return
        if msg.command == 'ping':
            msg.command = 'ping_ok'
            self.fdn.send(msg)
        elif msg.command == 'close':
            # Ground control has requested a close down
            self.fds.send(msg) # Notify spectrometer
            msg.command = 'close_ok' # Convert the message to a response
            self.fdn.send(msg) # Notify network process and ground control
            self.running = False
        elif msg.command == 'new_session' or msg.command == 'stop_session' or msg.command == 'set_gain':
            self.fds.send(msg) # No housekeeping necessary, pass directly to spectrometer
        else:
            # Unknown command received from network
             logging.warning('ctrl: unknown command: ' + msg.command)

```

7.1.3.4 def burn.burn.Burn.dispatch_spec_msg (*self, msg*)

Definition at line 102 of file burn.py.

```
    def dispatch_spec_msg(self, msg):
        # Message received from spectrometer, pass on to network
        self.fdn.send(msg)

```

7.1.3.5 def burn.burn.Burn.run (*self*)

```
Entry point for the main controller
```

Definition at line 57 of file burn.py.

```
    def run(self):
        """
        Entry point for the main controller
        """
        self.running = True

        # Wait for gps to start up
        logging.info('ctrl: warming up services')
        time.sleep(5)

        # Prepare file descriptors for selection
        inputs = [self.fdn, self.fds]

        # Main event loop
        while self.running:
            readable, _, _ = select.select(inputs, [], [])
            # Process readable file descriptors
            for s in readable:
                msg = s.recv()
                if s is self.fdn: # Message received from network
                    self.dispatch_net_msg(msg) # Handle network message
                elif s is self.fds: # Message received from spectrometer
                    self.dispatch_spec_msg(msg) # Handle spectrometer message

```

7.1.4 Member Data Documentation

7.1.4.1 burn.burn.Burn.fdn

Definition at line 40 of file burn.py.

7.1.4.2 burn.burn.Burn.fds

Definition at line 39 of file burn.py.

7.1.4.3 burn.burn.Burn.n

Definition at line 48 of file burn.py.

7.1.4.4 burn.burn.Burn.running

Definition at line 36 of file burn.py.

7.1.4.5 burn.burn.Burn.s

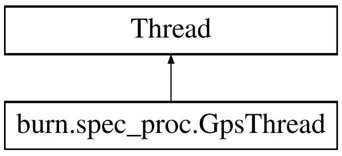
Definition at line 47 of file burn.py.

The documentation for this class was generated from the following file:

- /home/drb/dev/py/burn/burn.py

7.2 burn.spec_proc.GpsThread Class Reference

Inheritance diagram for burn.spec_proc.GpsThread:



Public Member Functions

- def __init__ (self, event)
- def run (self)
- def latitude (self)
- def epx (self)
- def longitude (self)
- def epy (self)
- def altitude (self)
- def epv (self)
- def speed (self)
- def eps (self)
- def time (self)

Public Attributes

- gpsd
- last_lat
- last_epx
- last_lon
- last_epy
- last_alt
- last_epv
- last_speed
- last_eps
- last_time

7.2.1 Detailed Description

```
Thread class to handle the gps driver
```

Definition at line 38 of file spec_proc.py.

7.2.2 Constructor & Destructor Documentation

7.2.2.1 def burn.spec_proc.GpsThread.__init__ (*self*, *event*)

```
Description:
    Initialize the gps thread
```

Definition at line 42 of file spec_proc.py.

```
    def __init__(self, event):
        """
        Description:
            Initialize the gps thread
        """
        threading.Thread.__init__(self)
        self._stopped = event
        self.gpsd = gps(mode=WATCH_ENABLE)
        self.last_lat = 0
        self.last_epx = 0
        self.last_lon = 0
        self.last_epy = 0
        self.last_alt = 0
        self.last_epv = 0
        self.last_speed = 0
        self.last_eps = 0
        self.last_time = ''

```

7.2.3 Member Function Documentation

7.2.3.1 def burn.spec_proc.GpsThread.altitude (*self*)

Definition at line 111 of file spec_proc.py.

```
    def altitude(self):
        return self.last_alt

```

7.2.3.2 def burn.spec_proc.GpsThread.eps (*self*)

Definition at line 123 of file spec_proc.py.

```
    def eps(self):
        return self.last_eps

```

7.2.3.3 def burn.spec_proc.GpsThread.epv (*self*)

Definition at line 115 of file spec_proc.py.

```
    def epv(self):
        return self.last_epv

```

7.2.3.4 def burn.spec_proc.GpsThread.epx (*self*)

Definition at line 99 of file spec_proc.py.

```
    def epx(self):
        return self.last_epx

```

7.2.3.5 def burn.spec_proc.GpsThread.epy (*self*)

Definition at line 107 of file spec_proc.py.

```
    def epy(self):
        return self.last_epy

```

7.2.3.6 def burn.spec_proc.GpsThread.latitude (*self*)

Definition at line 95 of file spec_proc.py.

```
    def latitude(self):
        return self.last_lat

```

7.2.3.7 def burn.spec_proc.GpsThread.longitude (*self*)

Definition at line 103 of file spec_proc.py.

```
    def longitude(self):
        return self.last_lon

```

7.2.3.8 def burn.spec_proc.GpsThread.run (*self*)

```
Description:
    Entry point for the gps thread
```

Definition at line 60 of file spec_proc.py.

```
    def run(self):
        """
        Description:
            Entry point for the gps thread
        """
        logging.info('gps: starting service')

        # Process any buffered gps signals every .3 seconds
        while not self._stopped.wait(0.3):

            # Update our last measurement until buffer is empty
            while self.gpsd.waiting():
                self.gpsd.next()
                if not math.isnan(self.gpsd.fix.latitude):
                    self.last_lat = self.gpsd.fix.latitude
                if not math.isnan(self.gpsd.fix.epx):
                    self.last_epx = self.gpsd.fix.epx
                if not math.isnan(self.gpsd.fix.longitude):
                    self.last_lon = self.gpsd.fix.longitude
                if not math.isnan(self.gpsd.fix.epy):
                    self.last_epy = self.gpsd.fix.epy
                if not math.isnan(self.gpsd.fix.altitude):
                    self.last_alt = self.gpsd.fix.altitude
                if not math.isnan(self.gpsd.fix.epv):
                    self.last_epv = self.gpsd.fix.epv
                if not math.isnan(self.gpsd.fix.speed):
                    self.last_speed = self.gpsd.fix.speed
                if not math.isnan(self.gpsd.fix.eps):
                    self.last_eps = self.gpsd.fix.eps
                if self.gpsd.utc != None and self.gpsd.utc != '':
                    self.last_time = self.gpsd.utc

        logging.info('gps: terminating')

```

7.2.3.9 def burn.spec_proc.GpsThread.speed (*self*)

Definition at line 119 of file spec_proc.py.

```
    def speed(self):
        return self.last_speed

```

7.2.3.10 def burn.spec_proc.GpsThread.time (*self*)

Definition at line 127 of file spec_proc.py.

```
    def time(self):
        return self.last_time

```

7.2.4 Member Data Documentation

7.2.4.1 burn.spec_proc.GpsThread.gpsd

Definition at line 49 of file spec_proc.py.

7.2.4.2 burn.spec_proc.GpsThread.last_alt

Definition at line 54 of file spec_proc.py.

7.2.4.3 burn.spec_proc.GpsThread.last_eps

Definition at line 57 of file spec_proc.py.

7.2.4.4 burn.spec_proc.GpsThread.last_epv

Definition at line 55 of file spec_proc.py.

7.2.4.5 burn.spec_proc.GpsThread.last_epx

Definition at line 51 of file spec_proc.py.

7.2.4.6 burn.spec_proc.GpsThread.last_epy

Definition at line 53 of file spec_proc.py.

7.2.4.7 burn.spec_proc.GpsThread.last_lat

Definition at line 50 of file spec_proc.py.

7.2.4.8 burn.spec_proc.GpsThread.last_lon

Definition at line 52 of file spec_proc.py.

7.2.4.9 burn.spec_proc.GpsThread.last_speed

Definition at line 56 of file spec_proc.py.

7.2.4.10 burn.spec_proc.GpsThread.last_time

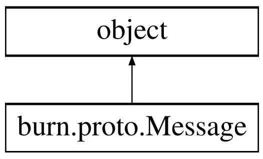
Definition at line 58 of file spec_proc.py.

The documentation for this class was generated from the following file:

- /home/drb/dev/py/burn/spec_proc.py

7.3 burn.proto.Message Class Reference

Inheritance diagram for burn.proto.Message:



Public Member Functions

- def __init__ (self, command='', arguments={})

Public Attributes

- command
- arguments

7.3.1 Detailed Description

```
Class used to store a protocol message
```

Definition at line 20 of file proto.py.

7.3.2 Constructor & Destructor Documentation

7.3.2.1 def burn.proto.Message.__init__ (*self*, *command* = '', *arguments* = {})

```
Initialize message
```

Definition at line 24 of file proto.py.

```
    def __init__(self, command='', arguments={}):
        """
        Initialize message
        """
        self.command = command
        self.arguments = arguments

```

7.3.3 Member Data Documentation

7.3.3.1 burn.proto.Message.arguments

Definition at line 29 of file proto.py.

7.3.3.2 burn.proto.Message.command

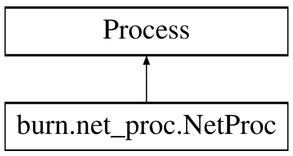
Definition at line 28 of file proto.py.

The documentation for this class was generated from the following file:

- /home/drb/dev/py/burn/proto.py

7.4 burn.net_proc.NetProc Class Reference

Inheritance diagram for burn.net_proc.NetProc:



Public Member Functions

- def __init__ (self, fd)
- def run (self)
- def dispatch_ctrl_msg (self, msg)
- def dispatch_net_msg (self)
- def is_running (self)

Public Attributes

- fd
- addr
- sock
- buffer

7.4.1 Detailed Description

Definition at line 28 of file net_proc.py.

7.4.2 Constructor & Destructor Documentation

7.4.2.1 def burn.net_proc.NetProc.__init__ (*self*, *fd*)

```
Description:
    Initialization of the net process
Arguments:
    fd - File descriptor to send and receive messages to/from controller
```

Definition at line 30 of file net_proc.py.

```
    def __init__(self, fd):
        """
        Description:
            Initialization of the net process
        Arguments:
            fd - File descriptor to send and receive messages to/from controller
        """
        Process.__init__(self)
        self.fd = fd
        setblocking(self.fd, 0)
        self._running = False
        self.conn, self.addr = None, None
        self.sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        self.sock.setblocking(0)
        self.buffer = ''
        try:
            self.sock.bind((HOST, PORT))
        except socket.error as e:
            logging.error('network: bind failed: ' + os.strerror(e.errno))

        self.sock.listen(5)
        logging.info('network: service listening')

```

7.4.3 Member Function Documentation

7.4.3.1 def burn.net_proc.NetProc.dispatch_ctrl_msg (*self*, *msg*)

```
Description:
    Serialize a controller message to a netstring and send it to ground control
Arguments:
    msg - The controller message to dispatch
```

Definition at line 109 of file net_proc.py.

```
    def dispatch_ctrl_msg(self, msg):
        """
        Description:
            Serialize a controller message to a netstring and send it to ground control
        Arguments:
            msg - The controller message to dispatch
        """
        if msg.command == 'close_ok': # main controller is closing
            self._running = False

        data = ''
        if msg.command == 'spectrum_ready': # Meta message
            # 'spectrum_ready' is a meta message indicating that a response message is stored on disk
            # The path to the file is stored in msg.arguments['filename']
            with open(msg.arguments["filename"]) as jfd:
                m = json.load(jfd) # Load json from file
            data = json.dumps(m)
        else: # Regular message
            data = json.dumps(msg.__dict__) # Convert object to json

        # Serialize the message into a netstring
        netstring = struct.pack("!I", len(data)) + data
        # Send the message over the network
        totlen, currlen = len(netstring), 0
        while True: # Continue until the full message is transferred
            l = self.conn.send(netstring[currlen:])
            if l == 0:
                inputs.remove(self.conn)
                self.conn.close()
                logging.info('network: connection broken from ' + self.addr[0])
                break
            currlen += l
            if currlen >= totlen:
                break

```

7.4.3.2 def burn.net_proc.NetProc.dispatch_net_msg (*self*)

```
Description:
    Convert received data to messages and pass them to the controller
```

Definition at line 144 of file net_proc.py.

```
    def dispatch_net_msg(self):
        """
        Description:
            Convert received data to messages and pass them to the controller
        """
        while True:
            if len(self.buffer) < 4:
                return
            # Extract message length
            msglen = struct.unpack("!I", self.buffer[0:4])[0]
            if len(self.buffer) < msglen+4:
                logging.info('network: buffer not ready')
                return
            # Extract rest of message and convert to object
            jmsg = json.loads(self.buffer[4:4+msglen])
            msg = Message(**jmsg)
            # Pass message to main controller
            self.fd.send(msg)
            # Update buffer
            self.buffer = self.buffer[4+msglen:]

```

7.4.3.3 def burn.net_proc.NetProc.is_running (*self*)

```
Description:
    Return wether the net process is still running
```

Definition at line 165 of file net_proc.py.

```
    def is_running(self):
        """
        Description:
            Return wether the net process is still running
        """
        return self._running

```

7.4.3.4 def burn.net_proc.NetProc.run (*self*)

```
Description:
    Entry point for the net process
```

Definition at line 53 of file net_proc.py.

```
53      def run(self):
54          """
55          Description:
56              Entry point for the net process
57          """
58          logging.info('network: starting service')
59          self._running = True
60          # Prepare sockets and file descriptors
61          inputs = [self.fd, self.sock]
62
63          # Start select event loop
64          while(self._running):
65              readable, _, _ = select.select(inputs, [], [])
66
67              for s in readable: # Handle reads
68                  if s is self.sock: # Incoming connection on listening socket
69                      # We only allow one connection at a time (TODO)
70                      self.conn, self.addr = s.accept()
71                      self.conn.setblocking(0)
72                      inputs.append(self.conn)
73                      self.buffer = ''
74                      logging.info('network: connection received from ' + self.
      addr[0])
75
76                  elif s is self.fd: # Incoming message from main controller
77                      self.dispatch_ctrl_msg(s.recv())
78
79                  else: # Incoming data from existing connection
80                      try:
81                          data = s.recv(1024)
82                      except socket.error as e:
83                          if e.errno == errno.ECONNRESET:
84                              # Unexpected disconnect from client
85                              inputs.remove(s)
86                              s.close()
87                          logging.error('network: ' + self.addr[0] + ': ' + os.strerror(e.errno))
88                          continue
89                      if not data or data == '':
90                          # Unexpected disconnect from client
91                          inputs.remove(s)
92                          s.close()
93                          logging.error('network: connection lost')
94                          continue
95                      else:
96                          # Data successfully received, store in buffer
97                          self.buffer += data
98                          self.dispatch_net_msg()
99
100         # Close active connections
101         if self.conn is not None:
102             self.conn.close()
103         if self.sock is not None:
104             self.sock.close()
105         self.fd.close()
106
107         logging.info('network: terminating')
108
```

7.4.4 Member Data Documentation

7.4.4.1 burn.net_proc.NetProc.addr

Definition at line 41 of file net_proc.py.

7.4.4.2 burn.net_proc.NetProc.buffer

Definition at line 44 of file net_proc.py.

7.4.4.3 burn.net_proc.NetProc.fd

Definition at line 38 of file net_proc.py.

7.4.4.4 burn.net_proc.NetProc.sock

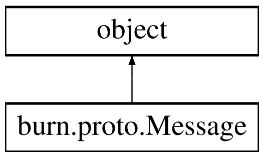
Definition at line 42 of file net_proc.py.

The documentation for this class was generated from the following file:

- /home/drb/dev/py/burn/net_proc.py

7.5 object Class Reference

Inheritance diagram for object:

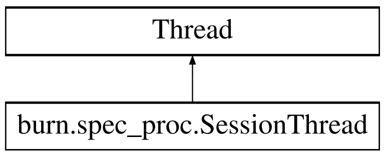


The documentation for this class was generated from the following file:

- /home/drb/dev/py/burn/proto.py

7.6 burn.spec_proc.SessionThread Class Reference

Inheritance diagram for burn.spec_proc.SessionThread:



Public Member Functions

- def __init__ (self, event, target, msg)
- def run (self)

7.6.1 Detailed Description

```
Thread class to govern a single session
```

Definition at line 130 of file spec_proc.py.

7.6.2 Constructor & Destructor Documentation

7.6.2.1 def burn.spec_proc.SessionThread.__init__ (*self, event, target, msg*)

```
Description:
    Initialize the session thread
Arguments:
    event - Event to notify exit
    target - Function running the detector
    msg - The session message containing info about this session
```

Definition at line 134 of file spec_proc.py.

```
    def __init__(self, event, target, msg):
        """
        Description:
            Initialize the session thread
        Arguments:
            event - Event to notify exit
            target - Function running the detector
            msg - The session message containing info about this session
        """
        threading.Thread.__init__(self)
        self._stopped = event
        self._target = target
        self._msg = msg

```

7.6.3 Member Function Documentation

7.6.3.1 def burn.spec_proc.SessionThread.run (*self*)

```
Description:
    Entry point for the session thread
```

Definition at line 148 of file spec_proc.py.

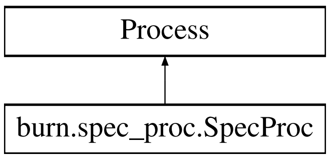
```
    def run(self):
        """
        Description:
            Entry point for the session thread
        """
        logging.info('session: starting')
        # Extract the session variables from the session message
        delay = float(self._msg.arguments["delay"]) # Time to eait between each spectrum
        iterations = int(self._msg.arguments["iterations"]) # Number of spectrums to take
        infinite = iterations == -1 # If iterations is -1, run forever
        index = 0 # Keep track of spectrums (spectrum id)

        while not self._stopped.wait(delay):
            if not infinite:
                # Exit when we reach a zero spectrum count
                iterations = iterations - 1
                if iterations < 0:
                    break
            # Run the detector
            self._target(self._msg, index)
            index = index + 1

        logging.info('session: terminating')

```

The documentation for this class was generated from the following file:

- /home/drb/dev/py/burn/spec_proc.py

7.7 burn.spec_proc.SpecProc Class Reference

Inheritance diagram for burn.spec_proc.SpecProc:



Public Member Functions

- def __init__ (self, fd)
- def run (self)
- def send_msg (self, msg)
- def dispatch (self, msg)
- def stabilize_probe (self, voltage, coarse_gain, fine_gain)
- def run_acquisition_once (self, req_msg, session_index)
- def reset_acquisition (self)
- def run_acquisition (self, msg, session_index)
- def save_acquisition (self, msg, session_index)
- def save_acquisition_as_chn (self, sd, msg, session_index)

Public Attributes

- fd
- running
- send_lock
- gps_stop
- gps_client
- group
- input
- dtb
- session_stop
- session

7.7.1 Detailed Description

```
Process class for handling the gps and spectrometry
```

Definition at line 172 of file spec_proc.py.

7.7.2 Constructor & Destructor Documentation

7.7.2.1 def burn.spec_proc.SpecProc.__init__ (*self*, *fd*)

```
Description:
    Initialize the spectrometer process
Arguments:
    fd - file descriptor to pass and receive messages to/from controller
```

Definition at line 176 of file spec_proc.py.

```
    def __init__(self, fd):
        """
        Description:
            Initialize the spectrometer process
        Arguments:
            fd - file descriptor to pass and receive messages to/from controller
        """
        Process.__init__(self)
        self.fd = fd
        self.running = False
        self.send_lock = threading.Lock() # Lock used to syncronize sending of messages to
    controller
        self.gps_stop = threading.Event() # Event used to notify gps thread
        self.gps_client = GpsThread(self.gps_stop) # Create the gps thread
        # Initalize the detector
        self.group = 1
        self.input = 1
        self.dtb = DeviceFactory.createInstance(DeviceFactory.DeviceInterface.IDevice)
        self.dtb.open("", Utilities.getLynxAddress())
        logging.info('spec: using device ' + self.dtb.getParameter(ParameterCodes.Network_MachineName, 0))
        self.dtb.lock("administrator", "password", self.input)

```

7.7.3 Member Function Documentation

7.7.3.1 def burn.spec_proc.SpecProc.dispatch (*self*, *msg*)

```
Description:
    Handle a message received from controller
Arguments:
    msg - The message received
```

Definition at line 234 of file spec_proc.py.

```
    def dispatch(self, msg):
        """
        Description:
            Handle a message received from controller
        Arguments:
            msg - The message received
        """
        if msg.command == 'set_gain': # Set the gain parameters for the detector
            voltage = msg.arguments["voltage"]
            coarse = msg.arguments["coarse_gain"]
            fine = msg.arguments["fine_gain"]
            self.stabilize_probe(voltage, coarse, fine)
            logging.info('spec: gain has been set')
            msg.command = 'set_gain_ok' # Notify ground control that gain has been set
            self.send_msg(msg)
        elif msg.command == 'close': # Controller wants us to close down
            self.running = False
        elif msg.command == 'new_session': # Start a new session
            msg.command = 'new_session_ok'
            self.send_msg(msg)
            self.session_stop = threading.Event()
            self.session = SessionThread(self.session_stop, self.
   run_acquisition_once, msg)
            self.session.start()
        elif msg.command == 'stop_session': # Stop any running sessions
            if not self.session_stop.isSet():
                self.session_stop.set()
                self.session.join()
                logging.info('spec: session stopped')
            msg.command = 'stop_session_ok' # Notify ground control that we have stopped any sessions
            self.send_msg(msg)
        else:
            # Unknown command received from controller
            logging.warning('spec: unknown command ' + cmd.command)

```

7.7.3.2 def burn.spec_proc.SpecProc.reset_acquisition (*self*)

```
Description:
    Reset and initialize the detector
```

Definition at line 340 of file spec_proc.py.

```
    def reset_acquisition(self):
        """
        Description:
            Reset and initialize the detector
        """
        #Disable all acquisition
        Utilities.disableAcquisition(self.dtb, self.input)
        #Set the acquisition mode. The Only Available Spectral in Osprey is Pha = 0
        self.dtb.setParameter(ParameterCodes.Input_Mode, 0, self.input)
        #Setup presets
        self.dtb.setParameter(ParameterCodes.Preset_Options, 1, self.input)
        #Clear data and time
        self.dtb.control(CommandCodes.Clear, self.input)
        #Set the current memory group
        self.dtb.setParameter(ParameterCodes.Input_CurrentGroup, self.group, self.
  input)

```

7.7.3.3 def burn.spec_proc.SpecProc.run (*self*)

```
Description:
    Entry point for the spectrometer process
```

Definition at line 197 of file spec_proc.py.

```
    def run(self):
        """
        Description:
            Entry point for the spectrometer process
        """
        logging.info('spec: staring service')
        self.running = True

        self.gps_client.start() # Start the gps

        # Event loop
        while(self.running):
            if self.fd.poll():
                self.dispatch(self.fd.recv()) # Handle messages from the controller

        # Cleanup and exit
        self.fd.close()
        self.gps_stop.set()
        self.gps_client.join()
        logging.info('spec: GPS client stopped')
        logging.info('spec: terminating')

```

7.7.3.4 def burn.spec_proc.SpecProc.run_acquisition (*self*, *msg*, *session_index*)

```
Description:
    Run the detector
Arguments:
    msg - The response message
    session_index - The sequence number in current session
```

Definition at line 356 of file spec_proc.py.

```
    def run_acquisition(self, msg, session_index):
        """
        Description:
            Run the detector
        Arguments:
            msg - The response message
            session_index - The sequence number in current session
        """
        # Setup presets
        livetime = float(msg.arguments["livetime"])
        self.dtb.setParameter(ParameterCodes.Preset_Live, livetime, self.input)
        # Clear data and time
        self.dtb.control(CommandCodes.Clear, self.input)
        # Start the acquisition
        self.dtb.control(CommandCodes.Start, self.input)
        while True:
            sd = self.dtb.getSpectralData(self.input, self.group)
            if ((0 == (StatusBits.Busy & sd.getStatus())) and (0 == (StatusBits.Waiting & sd.getStatus())))
    :
                break
            time.sleep(.1)

        # Extract last spectrum from detector and prepare parameters
        chans = sd.getSpectrum().getCounts()
        total_count = 0
        channel_string = ''
        for ch in chans:
            total_count += ch
            channel_string += str(ch) + ' '

        # Add spectrum data to the response message
        msg.arguments["channels"] = channel_string.strip()
        msg.arguments["channel_count"] = len(chans)
        msg.arguments["uncorrected_total_count"] = total_count
        msg.arguments["livetime"] = sd.getLiveTime()
        msg.arguments["realtime"] = sd.getRealTime()
        msg.arguments["computational_limit"] = sd.getComputationalValue()
        msg.arguments["spectral_input"] = sd.getInput()
        msg.arguments["spectral_group"] = sd.getGroup()
        msg.arguments["spectral_status"] = Utilities.getStatusDescription(sd.getStatus())

```

7.7.3.5 def burn.spec_proc.SpecProc.run_acquisition_once (*self*, *req_msg*, *session_index*)

```
Description:
    Gather info from gps and detector
Arguments:
    req_msg - The session message
    session_index - The sequence number in current session
```

Definition at line 293 of file spec_proc.py.

```
    def run_acquisition_once(self, req_msg, session_index):
        """
        Description:
            Gather info from gps and detector
        Arguments:
            req_msg - The session message
            session_index - The sequence number in current session
        """
        # Prepare the response message
        resp_msg = copy.deepcopy(req_msg)
        resp_msg.command = 'spectrum'
        resp_msg.arguments['session_index'] = session_index

        # Reset detector
        self.reset_acquisition()

        # Gather gps info before running the detector
        resp_msg.arguments['latitude_start'] = self.gps_client.latitude
        resp_msg.arguments['latitude_start_err'] = self.gps_client.epx
        resp_msg.arguments['longitude_start'] = self.gps_client.longitude
        resp_msg.arguments['longitude_start_err'] = self.gps_client.epy
        resp_msg.arguments['altitude_start'] = self.gps_client.altitude
        resp_msg.arguments['altitude_start_err'] = self.gps_client.epv
        resp_msg.arguments['gps_speed_start'] = self.gps_client.speed
        resp_msg.arguments['gps_speed_start_err'] = self.gps_client.eps
        resp_msg.arguments['gps_time_start'] = self.gps_client.time

        # Run the detector
        self.run_acquisition(resp_msg, session_index)

        # Gather gps info after running the detector
        resp_msg.arguments['gps_time_end'] = self.gps_client.time
        resp_msg.arguments['latitude_end'] = self.gps_client.latitude
        resp_msg.arguments['latitude_end_err'] = self.gps_client.epx
        resp_msg.arguments['longitude_end'] = self.gps_client.longitude
        resp_msg.arguments['longitude_end_err'] = self.gps_client.epy
        resp_msg.arguments['altitude_end'] = self.gps_client.altitude
        resp_msg.arguments['altitude_end_err'] = self.gps_client.epv
        resp_msg.arguments['gps_speed_end'] = self.gps_client.speed
        resp_msg.arguments['gps_speed_end_err'] = self.gps_client.eps

        # Save acquisition to file and send a meta message to controller
        fn = self.save_acquisition(resp_msg, session_index)
        m = Message('spectrum_ready')
        m.arguments["filename"] = fn
        self.send_msg(m)
```

7.7.3.6 def burn.spec_proc.SpecProc.save_acquisition (*self*, *msg*, *session_index*)

```
Description:
    Save the gps and specter data to file (json format)
Arguments:
    msg - The response message
    session_index - The sequence number in current session
```

Definition at line 396 of file spec_proc.py.

```
    def save_acquisition(self, msg, session_index):
        """
        Description:
            Save the gps and specter data to file (json format)
        Arguments:
            msg - The response message
            session_index - The sequence number in current session
        """
        # Build the path to store the response message
        session_name = msg.arguments['session_name']
        session_dir = os.path.expanduser("~/ashes/") + session_name
        if not os.path.isdir(session_dir):
            os.makedirs(session_dir, 0777)
        fname = session_dir + os.path.sep + str(session_index) + ".json"
        # Store the response message to file
        with open(fname, "w") as f:
            json.dump(msg.__dict__, f)
        return fname
```

7.7.3.7 def burn.spec_proc.SpecProc.save_acquisition_as_chn (*self*, *sd*, *msg*, *session_index*)

```
Description:
    Save the the specter data to file (chn format)
Arguments:
    sd - Spectrum data
    msg - The session message
    session_index - The sequence number in current session
```

Definition at line 415 of file spec_proc.py.

```
    def save_acquisition_as_chn(self, sd, msg, session_index):
        """
        Description:
            Save the the specter data to file (chn format)
        Arguments:
            sd - Spectrum data
            msg - The session message
            session_index - The sequence number in current session
        """
        session_name = msg.arguments['session_name']
        session_dir = os.path.expanduser("~/ashes/") + session_name
        if not os.path.isdir(session_dir):
            os.makedirs(session_dir, 0777)
        chans = sd.getSpectrum().getCounts()
        mca, sec, rt, lt, dat, tim, off, nc = 1, 0, sd.getRealTime(), sd.getLiveTime(), "07DEC151", "0707",
   0, len(chans) # FIXME
        hdr = pack("hhhhii8s4shh", -1, mca, 1, sec, rt, lt, dat, tim, off, nc)
        with open(session_dir + os.path.sep + str(session_index) + ".chn", "w+b") as f:
            f.write(hdr)
            int_array = array('L', chans)
            int_array.tofile(f)

```

7.7.3.8 def burn.spec_proc.SpecProc.send_msg (*self*, *msg*)

```
Description:
    Function to safely pass messages to controller
Arguments:
    msg - The message to pass
```

Definition at line 219 of file spec_proc.py.

```
    def send_msg(self, msg):
        """
        Description:
            Function to safely pass messages to controller
        Arguments:
            msg - The message to pass
        """
        self.send_lock.acquire()
        try:
            self.fd.send(msg)
        except:
            logging.error('spec: send exception: ' + sys.exc_info()[0])
        finally:
            self.send_lock.release()

```

7.7.3.9 def burn.spec_proc.SpecProc.stabilize_probe (*self*, *voltage*, *coarse_gain*, *fine_gain*)

```
Description:
    Set gain parameters for the detector
Arguments:
    voltage - The voltage level
    coarse_gain - The coarse gain level
    fine_gain - The fine gain level
```

Definition at line 268 of file spec_proc.py.

```
    def stabilize_probe(self, voltage, coarse_gain, fine_gain):
        """
        Description:
            Set gain parameters for the detector
        Arguments:
            voltage - The voltage level
            coarse_gain - The coarse gain level
            fine_gain - The fine gain level
        """
        # Osprey API constants
        Stabilized_Probe_Bussy = 0x00080000
        Stabilized_Probe_OK = 0x00100000
        dtb_probe_type = self.dtb.getParameter(ParameterCodes.Input_Status, self.
    input)
        # Set voltage
        if((dtb_probe_type & Stabilized_Probe_OK) != Stabilized_Probe_OK):
            self.dtb.setParameter(ParameterCodes.Input_Voltage, int(voltage), self.
    input)
            self.dtb.setParameter(ParameterCodes.Input_VoltageStatus, True, self.
    input)
            # Wait till ramping is complete
            logging.info('spec: ramping HVPS...')
            while(self.dtb.getParameter(ParameterCodes.Input_VoltageRamping, self.
    input) is True):
                time.sleep(.4)
        # Set coarse and fine gain
        self.dtb.setParameter(ParameterCodes.Input_CoarseGain, float(coarse_gain), self.
    input) # [1.0, 2.0, 4.0, 8.0]
        self.dtb.setParameter(ParameterCodes.Input_FineGain, float(fine_gain), self.
    input) # [1.0, 5.0]

```

7.7.4 Member Data Documentation

7.7.4.1 burn.spec_proc.SpecProc.dtb

Definition at line 192 of file spec_proc.py.

7.7.4.2 burn.spec_proc.SpecProc.fd

Definition at line 184 of file spec_proc.py.

7.7.4.3 burn.spec_proc.SpecProc.gps_client

Definition at line 188 of file spec_proc.py.

7.7.4.4 burn.spec_proc.SpecProc.gps_stop

Definition at line 187 of file spec_proc.py.

7.7.4.5 burn.spec_proc.SpecProc.group

Definition at line 190 of file spec_proc.py.

7.7.4.6 burn.spec_proc.SpecProc.input

Definition at line 191 of file spec_proc.py.

7.7.4.7 burn.spec_proc.SpecProc.running

Definition at line 185 of file spec_proc.py.

7.7.4.8 burn.spec_proc.SpecProc.send_lock

Definition at line 186 of file spec_proc.py.

7.7.4.9 burn.spec_proc.SpecProc.session

Definition at line 255 of file spec_proc.py.

7.7.4.10 burn.spec_proc.SpecProc.session_stop

Definition at line 254 of file spec_proc.py.

The documentation for this class was generated from the following file:

- /home/drb/dev/py/burn/spec_proc.py

Chapter 8

File Documentation

8.1 /home/drb/dev/py/burn/__init__.py File Reference

Namespaces

- burn

8.2 /home/drb/dev/py/burn/burn.py File Reference

Classes

- class burn.burn.Burn

Namespaces

- burn.burn

Variables

- burn.burn.filename
- burn.burn.level

8.3 /home/drb/dev/py/burn/helpers.py File Reference

Namespaces

- burn.helpers

Functions

- def burn.helpers.setblocking (fd, state)

8.4 /home/drb/dev/py/burn/net_proc.py File Reference

Classes

- class burn.net_proc.NetProc

Namespaces

- burn.net_proc

Variables

- string burn.net_proc.HOST = ''
- int burn.net_proc.PORT = 7000

8.5 /home/drb/dev/py/burn/proto.py File Reference

Classes

- class burn.proto.Message

Namespaces

- burn.proto

8.6 /home/drb/dev/py/burn/README.md File Reference

8.7 /home/drb/dev/py/burn/spec_proc.py File Reference

Classes

- class burn.spec_proc.GpsThread
- class burn.spec_proc.SessionThread
- class burn.spec_proc.SpecProc

Namespaces

- burn.spec_proc

Index